```
from keras.datasets import cifar10
import numpy

(x_train_ori, y_train_ori), (x_test_ori, y_test_ori) = cifar10.load_data()

print('shape of x_train: ' + str(x_train_ori.shape))
print('shape of y_train: ' + str(y_train_ori.shape))
print('shape of x_test: ' + str(x_test_ori.shape))
print('shape of y_test: ' + str(y_test_ori.shape))
print('number of classes: ' + str(numpy.max(y_train_ori) - numpy.min(y_train_ori) + 1))

```

```
Using TensorFlow backend.
shape of x_train: (50000, 32, 32, 3)
shape of y_train: (50000, 1)
shape of x_test: (10000, 32, 32, 3)
shape of y_test: (10000, 1)
number of classes: 10
```

```
x_train = x_train_ori/255
x_test = x_test_ori/255
print(x_train, x_test)
```

```
def to_one_hot(y, num_class=10):
    res = []
    for ys in y:
        code = [0]*num_class
        code[ys[0]] = 1
        res.append(code)
    return numpy.asarray(res)

y_train_vec = to_one_hot(y_train_ori)
y_test_vec = to_one_hot(y_test_ori)

print('Shape of y_train_vec: ' + str(y_train_vec.shape))
print('Shape of y_test_vec: ' + str(y_test_vec.shape))

print(y_train_ori[0])
print(y_train_vec[0])
```

```
Shape of y_train_vec: (50000, 10)
Shape of y_test_vec: (10000, 10)
[6]
[0 0 0 0 0 0 1 0 0 0]
```

```
rand_indices = numpy.random.permutation(50000)
train_indices = rand_indices[0:40000]
valid_indices = rand_indices[40000:50000]

x_val = x_train[valid_indices, :]
y_val = y_train_vec[valid_indices, :]

x_tr = x_train[train_indices, :]
y_tr = y_train_vec[train_indices, :]

print('Shape of x_tr: ' + str(x_tr.shape))
print('Shape of y_tr: ' + str(y_tr.shape))
print('Shape of x_val: ' + str(x_val.shape))
print('Shape of y_val: ' + str(y_val.shape))
```

```
Shape of x_tr: (40000, 32, 32, 3)
Shape of y_tr: (40000, 10)
Shape of x_val: (10000, 32, 32, 3)
Shape of y_val: (10000, 10)
```

```
batch_size = 32
epochs = 100
num_classes = 10
```

```
from keras import optimizers
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization,Activa
from keras.models import Sequential

model = Sequential()
model.add(Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:]))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

opt = optimizers.rmsprop(lr=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

model.fit(x_train, y_train_vec,
              batch_size=batch_size,
              epochs=epochs,
              shuffle=True)

```

```
score = model.evaluate(x_val, y_val, verbose=1)
print('Training loss: {0:.4f}\nTraining accuracy:  {1:.4f}'.format(*score))
```

```
import matplotlib.pyplot as plt
%matplotlib inline

acc = history.history['acc']
val_acc = history.history['val_acc']

es = range(len(acc))

plt.plot(es, acc, 'bo', label='Training acc')
plt.plot(es, val_acc, 'r', label='Validation acc')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        zca_epsilon=1e-06,
        rotation_range=0,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.,
        zoom_range=0.,
        channel_shift_range=0.,
        fill_mode='nearest',
        cval=0.,
        horizontal_flip=True,
        vertical_flip=False,
        rescale=None,
        preprocessing_function=None,
        data_format=None,
        validation_split=0.0)

datagen.fit(x_train)
```

```
model.fit_generator(datagen.flow(x_train, y_train_vec, batch_size=batch_size),
                    steps_per_epoch=x_train.shape[0] // batch_size,
                    epochs=epochs)
```

```
loss_and_acc = model.evaluate(x_test, y_test_vec)
print('loss = ' + str(loss_and_acc[0]))
print('accuracy = ' + str(loss_and_acc[1]))
```

Repeat Augmentation

```
baseMapNum = 32
weight_decay = 1e-4
```

```
from keras import regularizers
from keras import optimizers
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization,Activa
from keras.models import Sequential

model2 = Sequential()
model2.add(Conv2D(baseMapNum, (3,3), padding='same', kernel_regularizer=regularizers.l2
model2.add(Activation('relu'))
model2.add(BatchNormalization())
model2.add(Conv2D(baseMapNum, (3,3), padding='same', kernel_regularizer=regularizers.l2
model2.add(Activation('relu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Dropout(0.2))

model2.add(Conv2D(2*baseMapNum, (3,3), padding='same', kernel_regularizer=regularizers.
model2.add(Activation('relu'))
```

```
model2.add(BatchNormalization())
model2.add(Conv2D(2*baseMapNum, (3,3), padding='same', kernel_regularizer=regularizers.
model2.add(Activation('relu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Dropout(0.3))

model2.add(Conv2D(4*baseMapNum, (3,3), padding='same', kernel_regularizer=regularizers.
model2.add(Activation('relu'))
model2.add(BatchNormalization())
model2.add(Conv2D(4*baseMapNum, (3,3), padding='same', kernel_regularizer=regularizers.
model2.add(Activation('relu'))
model2.add(BatchNormalization())
model2.add(MaxPooling2D(pool_size=(2,2)))
model2.add(Dropout(0.4))

model2.add(Flatten())
model2.add(Dense(num_classes, activation='softmax'))

model2.summary()
```

```
from keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=False
    )
datagen.fit(x_train)
```

```
batch_size = 64
epochs=25
opt_rms = optimizers.rmsprop(lr=0.001,decay=1e-6)
model2.compile(loss='categorical_crossentropy',
        optimizer=opt_rms,
        metrics=['accuracy'])
model2.fit_generator(datagen.flow(x_train, y_train_vec, batch_size=batch_size),steps_pe
model2.save_weights('cifar10_normal_rms_ep75.h5')

opt_rms = optimizers.rmsprop(lr=0.0005,decay=1e-6)
model2.compile(loss='categorical_crossentropy',
        optimizer=opt_rms,
        metrics=['accuracy'])
model2.fit_generator(datagen.flow(x_train, y_train_vec, batch_size=batch_size),steps_pe
model2.save_weights('cifar10_normal_rms_ep100.h5')

opt_rms = optimizers.rmsprop(lr=0.0003,decay=1e-6)
model2.compile(loss='categorical_crossentropy',
        optimizer=opt_rms,
        metrics=['accuracy'])
model2.fit_generator(datagen.flow(x_train, y_train_vec, batch_size=batch_size),steps_pe
model2.save_weights('cifar10_normal_rms_ep125.h5')

```

```
loss_and_acc = model2.evaluate(x_test, y_test_vec)
print('loss = ' + str(loss_and_acc[0]))
print('accuracy = ' + str(loss_and_acc[1]))
```

```
10000/10000 [==============================] - 3s 291us/step
loss = 0.545775131225586
accuracy = 0.8662
```

```
score = model2.evaluate(x_val, y_val, verbose=1)
print('Training loss: {0:.4f}\nTraining accuracy:  {1:.4f}'.format(*score))
```

```
10000/10000 [==============================] - 3s 259us/step
Training loss: 0.4199
Training accuracy:  0.8983
```